REMARKS

INTRODUCTION:

Claims 13-15, 17-27 and 29-43 have been cancelled without prejudice or disclaimer and claim 1 has been amended. Support for the claim amendment may be found at least at paragraph [0032] of the present application and therefore no new matter has been added.

Claims 1-3 and 5-12 are pending and under consideration. Claim 1 is an independent claim. Applicant requests reconsideration and allowance of the present application in view of the amendments above and the following remarks.

REJECTION UNDER 35 U.S.C. §101:

Claims 43 stands rejected under 35 U.S.C. §101 as being directed to non-statutory subject matter. The claim has been cancelled and therefore the rejection should be withdrawn.

REJECTIONS UNDER 35 USC 102:

Claim 1-43 stand rejected under 35 U.S.C. 102(e) as being anticipated by Cato et al., U.S. Patent Application No. 2003/0120928 ("Cato"). The rejection is respectfully traversed.

Amended independent claim 1 recites at least the following:

> generating a plurality of metadata fragment data by partitioning metadata to be transmitted based upon a predetermined semantic unit

Cato fails to suggest or disclose at least the above-claimed features. Cato is directed to a digital rights management system for securely associating rules for authorized use with digital information. More specifically, Cato discusses the distribution "of digital information protected by a cryptographically secure container (CSC) by providing an unencrypted portion of a data structure that at least in part describes the rules associated with the protected digital information or content. In some embodiments the unencrypted rules description or rules metadata may be part of the secure container data structure..." (*Cato* at paragraph [0076]).

The current Office Action asserts Cato suggests the above-recited features because "the examiner assumes that the digital content will have to parse into smaller chunks for delivery which is equivalent to generating a plurality of metadata fragment data by partitioning metadata to be transmitted and metadata fragment data that are predetermined semantic units." Applicant respectfully asserts that a rejection may not be based on such an "assumption." MPEP § 2131 states that

> "[t]o anticipate a claim, the reference must teach every element of the claim." A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference."

Moreover, to serve as an anticipating reference, the reference must enable that which it is asserted to anticipate. *Amgen, Inc. v. Hoechst Marion Roussel, Inc.*, 314 F.3d 1313, 1354, 65 USPQ2d 1385, 1416 (Fed. Cir. 2003). The Office Action has failed to demonstrate that the above-recited features are expressly or inherently set forth in Cato. Further, the Office Action has failed to demonstrate that Cato is an enabling reference as required above. It is not apparent to Applicant how Cato can teach that which it does not even describe.

The Office Action relies on the fact that Cato "also uses the TCP/IP transport layers to deliver authorized digital content" as the basis for the assumption that "Cato must parse metadata in a way equivalent to the above-recited claim language." Again, Applicant respectfully disagrees because even assuming for the sake of argument that Cato's use of TCP/IP results in the parsing of data, Cato's parsing of data is not limited to parsing of metadata but is instead directed to the parsing of data and metadata combined. Applicant asserts that the indiscriminate parsing of a combination of data and metadata does not suggest or disclose "generating a plurality of metadata fragment data," as recited above.

Further, TCP/IP does not suggest "partitioning metadata to be transmitted based upon a predetermined semantic unit." In the TCP/IP model, the transport layer uses statistical multiplexing techniques, such as packet switching, for delivering data (*See* Wikipedia, Transport Layer description, page 1 of 4, third paragraph – attached). Packet switching in the TCP/IP model typically uses the TCP transport protocol, which breaks data into chunks called transport segments. A TCP transport segment consists of two parts including a header section and data section. (*See* Wikipedia, Transmission Control Protocol description, TCP segment structure section, page 10 of 14 – attached). The structure of the transport segment is created according to a need for efficiency in transport.

In contrast, the word semantic is defined by the Merriam-Webster online dictionary as "relating to meaning in language." Accordingly, Applicant asserts that TCP/IP's chunking of data into transport segments for delivery is not "based upon a predetermined semantic unit" because the data is chunked into a transport segment based on a need for efficiency in transport and not chunked into a unit having meaning in language.

Amended independent claim 1 further recites at least the following:

> transmitting the selected metadata fragment data and the metadata-related information with data format information indicating a type of the selected metadata fragment data

Cato fails to suggest or disclose at least the above-claimed features. The Office Action asserts that Cato illustrates and describes the above-recited features at FIGS. 6 and 8 and paragraphs [0089], [0091], [0093], [0096], and [0099]. Applicant respectfully disagrees. The paragraphs cited in the Office Action describe "a P2P file search/transfer protocol" in communication with various layers including a "functional layer," an "API layer" and an "application layer." The only portion of the cited text that describes transmitting data is paragraph [0091], which describes creating a partially encrypted stream, including streamed media files, that is sent to a recipient. The paragraph further describes sending a first SSC to the recipient that includes a key(s) and rule information such that the player could decrypt and play a second SSC that includes the media file. However, nowhere in paragraph [0091] or in the other cited paragraphs does Cato describe "transmitting... data format information indicating a type of the selected metadata fragment data."

In contrast, in a non-limiting embodiment of the present application, "data format information specifying a format type of the selected metadata fragment data, for example, by indicating whether the format of the selected metadata fragment data is binary XML or text XML, is generated and then transmitted to the client" (see, for example, par. [0034]).

Accordingly, Applicant respectfully submits that amended independent claim 1 patentably distinguishes over Cato, and should be allowable for at least the above-mentioned reasons. Regarding the rejection of claims 2-3 and 5-12, these claims depend directly or indirectly on independent claim 1 and are therefore believed to be allowable for at least the reasons noted above.

REQUEST FOR INTERVIEW BEFORE FIRST OFFICE ACTION:

Applicants respectfully request the Examiner contact the undersigned attorney to discuss the pending claims before issuance of a First Office Action. Applicants believe that a more thorough review of the pending claims will be helpful to further prosecution.

CONCLUSION:

There being no further outstanding objections or rejections, it is submitted that the application is in condition for allowance. An early action to that effect is courteously solicited.

Finally, if there are any formal matters remaining after this response, the Examiner is requested to telephone the undersigned to attend to these matters.

If there are any additional fees associated with filing of this Amendment, please charge the same to our Deposit Account No. 19-3935.

Respectfully submitted,

STAAS & HALSEY LLP

Date: Oct. 23, 2007

By: _____

David J. Cutitta
Registration No. 52,790

1201 New York Avenue, N.W., 7th Floor
Washington, D.C. 20005
Telephone: (202) 434-1500
Facsimile: (202) 434-1501

Transport layer

From Wikipedia, the free encyclopedia

In computing and telecommunications, the **transport layer** is the second highest layer in the four and five layer TCP/IP reference models, where it responds to service requests from the application layer and issues service requests to the Internet layer. It is also the name of layer four of the seven layer OSI model, where it responds to service requests from the session layer and issues service requests to the network layer. The definitions of the transport layer are slightly different in these two models. This article primarily refers to the TCP/IP model. See also the OSI model definition of the transport layer.

OSI Model
7 Application layer
6 Presentation layer
5 Session layer
4 **Transport layer**
3 Network layer
2 Data link layer
▪ LLC sublayer ▪ MAC sublayer
1 Physical layer

A **transport protocol** is a protocol on the transport layer. The two most widely used transport protocols on the Internet are the connection oriented TCP (Transmission Control Protocol), and UDP (User Datagram Protocol). TCP is the more complicated and most common. Other options are the Datagram Congestion Control Protocol (DCCP) and Stream Control Transmission Protocol (SCTP).

The five-layer TCP/IP model
5. Application layer
DHCP • DNS • FTP • Gopher • HTTP • IMAP4 • IRC • NNTP • XMPP • POP3 • SIP • SMTP • SNMP • SSH • TELNET • RPC • RTP • RTCP • RTSP • TLS/SSL • SDP • SOAP • BGP • PPTP • L2TP • GTP • STUN • NTP • ...
4. Transport layer
TCP • UDP • DCCP • SCTP • RSVP • ...
3. Network/Internet Layer
IP (IPv4 • IPv6) • IGMP • ICMP • OSPF • ISIS • IPsec • ARP • RARP • RIP • ...
2. Data link layer
802.11 • ATM • DTM • Token Ring • Ethernet • FDDI • Frame Relay • GPRS • EVDO • HSPA • HDLC • PPP • ...
1. Physical layer
Ethernet physical layer • ISDN • Modems • PLC • SONET/SDH • G.709 • Optical Fiber • WiFi • WiMAX • Coaxial Cable • Twisted Pair • ...

The transport layer is typically handled by processes in the host computer operational system, and not by routers and switches. The transport layer usually turns the unreliable and very basic service provided by the Network layer into a more powerful one.

In the TCP/IP model, the transport layer is responsible for delivering data to the appropriate application process on the host computers. This involves statistical multiplexing of data from different application processes, i.e. forming data packets, and adding source and destination port numbers in the header of each transport layer data packet. Together with the source and destination IP address, the port numbers constitutes a network socket, i.e. an identificiation address of the process-to-process communication. In the OSI model, this function is supported by the session layer.

Some transport layer protocols, for example TCP but not UDP, support virtual circuits, i.e. provide connection oriented communication over an underlying packet oriented datagram network. A byte-stream is delivered while hiding the packet mode communication for the application processes. This involves connection establishment, dividing of the data stream into packets called segments, segment numbering and reordering of out-of order data.

Finally, some transport layer protocols, for example TCP but not UDP, provides end-to-end reliable communication, i.e. error recovery by means of error detecting code and automatic repeat request (ARQ) protocol. The ARQ protocol also provides flow control, which may be combined with congestion avoidance.

UDP is a very simple service, and does not provide virtual circuits, nor reliable communication, leaving these to the application. The UDP packets are called datagrams rather than segments.

TCP is used for for example HTTP web browsing and email transfer. UDP may be used for multicasting and broadcasting, since retransmissions are not possible to a large amount of hosts. UDP typically gives higher throughput and shorter latency, and is therefor often used for realtime multimedia communication where packet loss occasionally can be accepted, for example IP-TV and IP-telephony, and for online computer games.

In many non-IP-based networks, for example X.25, Frame Relay and ATM, the connection oriented communication is implemented at network layer or data link layer rather than the transport layer. In X.25, in telephone network modems and in wireless communication systems, reliable node-to-node communication is implemented at lower protocol layers.

In the OSI/X.25 protocol suite, there are five classes of the OSI transport protocol, ranging from class 0 (which is also known as **TP0** and provides the least error recovery) to class 4 (which is also known as **TP4** and is designed for less reliable networks, similar to the Internet).

List of transport layer services

There is a long list of services that can be optionally provided by the transport layer. None of them are compulsory, because not all applications want all the services available. Some can be wasted overhead, or even counterproductive in some cases.

Connection-oriented
: This is normally easier to deal with than connection-less models, so where the Network layer only provides a connection-less service, often a connection-oriented service is built on top of that in the Transport layer.

Same Order Delivery
: The Network layer doesn't generally guarantee that packets of data will arrive in the same order that they were sent, but often this is a desirable feature, so the Transport layer provides it. The simplest way of doing this is to give each packet a number, and allow the receiver to reorder the packets.

Reliable Data
: Packets may be lost in routers, switches, bridges and hosts due to network congestion, when the packet queues are filled and the network nodes have to delete packets. Packets may be lost or corrupted in for example Ethernet due to interference and noise, since Ethernet does not retransmit

corrupt packets. Packets may be delivered in the wrong order by an underlying network. Some transport layer protocols, for example TCP, can fix this. By means of an error detection code, for example a checksum, the transport protocol may check that the data is not corrupted, and verify that by sending an ACK message to the sender. Automatic repeat request schemes may be used to retransmit lost or corrupted data. By introducing segment numbering in the transport layer packet headers, the packets can be sorted in order. Of course, error free is impossible, but it is possible to substantially reduce the numbers of undetected errors.

Flow Control

The amount of memory on a computer is limited, and without flow control a larger computer might flood a computer with so much information that it can't hold it all before dealing with it. Nowadays, this is not a big issue, as memory is cheap while bandwidth is comparatively expensive, but in earlier times it was more important. Flow control allows the receiver to say "Whoa!" before it is overwhelmed. Sometimes this is already provided by the network, but where it is not, the Transport layer may add it on.

Congestion avoidance

Network congestion occurs when a queue buffer of a network node is full and starts to drop packets. Automatic repeat request may keep the network in a congested state. This situation can be avoided by adding congestion avoidance to the flow control, including slow-start. This keeps the bandwidth consumption at a low level in the beginning of the transmission, or after packet retransmission.

Byte orientation

Rather than dealing with things on a packet-by-packet basis, the Transport layer may add the ability to view communication just as a stream of bytes. This is nicer to deal with than random packet sizes, however, it rarely matches the communication model which will normally be a sequence of messages of user defined sizes.

Ports

(Part of the transport layer in the TCP/IP model, but of the session layer in the OSI model) Ports are essentially ways to address multiple entities in the same location. For example, the first line of a postal address is a kind of port, and distinguishes between different occupants of the same house. Computer applications will each listen for information on their own ports, which is why you can use more than one network-based application at the same time.

Transport protocol comparison table

	UDP	TCP	DCCP	SCTP
Packet header size	8 Bytes	20 Bytes	Varies	12 Bytes + Variable Chunk Header
Transport layer packet entity	Datagram	Segment	Datagram	Datagram
Port numbering	Yes	Yes	Yes	Yes
Error detection	Optional	Yes	Yes	Yes
Reliability: Error recovery by automatic repeat request (ARQ)	No	Yes	No	Yes
Virtual circuits: Sequence numbering and reordering	No	Yes	Yes	Optional
Flow control	No	Yes	Yes	Yes
Congestion avoidance: Variable congestion window, slow start, time outs	No	Yes	Yes	Yes

Transmission Control Protocol

From Wikipedia, the free encyclopedia

The **Transmission Control Protocol** (**TCP**) is one of the core protocols of the Internet protocol suite. TCP provides reliable, in-order delivery of a stream of bytes, making it suitable for applications like file transfer and e-mail. It is so important in the Internet protocol suite that sometimes the entire suite is referred to as "the TCP/IP protocol suite."

The five-layer TCP/IP model

5. Application layer

DHCP · DNS · FTP · Gopher · HTTP · IMAP4 · IRC · NNTP · XMPP · POP3 · SIP · SMTP · SNMP · SSH · TELNET · RPC · RTCP · RTSP · TLS · SDP · SOAP · GTP · STUN · NTP · RIP · ...

4. Transport layer

TCP · UDP · DCCP · SCTP · RTP · RSVP · IGMP · ICMP · ICMPv6 · PPTP · ...

3. Network/Internet layer

IP (IPv4 · IPv6) · OSPF · IS-IS · BGP · IPsec · ARP · RARP · ...

2. Data link layer

802.11 · Wi-Fi · WiMAX · ATM · DTM · Token Ring · Ethernet · FDDI · Frame Relay · GPRS · EVDO · HSPA · HDLC · PPP · L2TP · ISDN · ...

1. Physical layer

Ethernet physical layer · Modems · PLC · SONET/SDH · G.709 · OFDM · Optical Fiber · Coaxial Cable · Twisted Pair · ...

Contents

Reason for TCP

The Internet Protocol (IP) works by exchanging groups of information called packets. Packets are short sequences of bytes consisting of a header and a body. The header describes the packet's destination,

which routers on the Internet use to pass the packet along, in generally the right direction, until it arrives at its final destination. The body contains the application data.

In cases of congestion, the IP protocol can discard packets, and, for efficiency reasons, two consecutive packets on the internet can take different routes to the destination. Then, the packets can arrive at the destination in the wrong order.

The TCP protocol's software libraries use the IP protocol and provides a simpler interface to applications by hiding most of the underlying packet structures, rearranging out-of-order packets, minimizing network congestion, and re-transmitting discarded packets. Thus, TCP very significantly simplifies the task of writing networked applications.

Applicability of TCP

TCP is used extensively by many of the Internet's most popular application protocols and resulting applications, including the World Wide Web, E-mail, File Transfer Protocol, Secure Shell, and some streaming media applications.

However, because TCP is optimized for accurate delivery rather than timely delivery, TCP sometimes incurs long delays while waiting for out-of-order messages or retransmissions of lost messages, and it is not particularly suitable for real-time applications such as Voice over IP. For such applications, protocols like the Real-time Transport Protocol (RTP) running over the User Datagram Protocol (UDP) are usually recommended instead[1].

Using TCP

Using TCP, applications on networked hosts can create *connections* to one another, over which they can exchange streams of data using Stream Sockets. TCP also distinguishes data for multiple connections by concurrent applications (*e.g.*, Web server and e-mail server) running on the same host.

In the Internet protocol suite, TCP is the intermediate layer between the Internet Protocol (IP) below it, and an application above it. Applications often need reliable pipe-like connections to each other, whereas the Internet Protocol does not provide such streams, but rather only best effort delivery (*i.e.*, unreliable packets). TCP does the task of the transport layer in the simplified OSI model of computer networks. The other main transport-level Internet protocols are UDP and SCTP.

Applications send streams of octets (8-bit bytes) to TCP for delivery through the network, and TCP divides the byte stream into appropriately sized segments (usually delineated by the maximum transmission unit (MTU) size of the data link layer of the network to which the computer is attached). TCP then passes the resulting packets to the Internet Protocol, for delivery through a network to the TCP module of the entity at the other end. TCP checks to make sure that no packets are lost by giving each packet a *sequence number*, which is also used to make sure that the data is delivered to the entity at the other end in the correct order. The TCP module at the far end sends back an *acknowledgment* for packets which have been successfully received; a timer at the sending TCP will cause a *timeout* if an acknowledgment is not received within a reasonable round-trip time (or RTT), and the (presumably) lost data will then be *re-transmitted*. The TCP checks that no bytes are corrupted by using a checksum; one is computed at the sender for each block of data before it is sent, and checked at the receiver.

Protocol operation

Unlike TCP's traditional counterpart, User Datagram Protocol, which can immediately start sending packets, TCP provides connections that need to be established before sending data. TCP connections have three phases:

1. connection establishment
2. data transfer
3. connection termination

Before describing these three phases, a note about the various states of a connection end-point or *Internet socket*:

1. LISTEN
2. SYN-SENT
3. SYN-RECEIVED
4. ESTABLISHED
5. FIN-WAIT-1
6. FIN-WAIT-2
7. CLOSE-WAIT
8. CLOSING
9. LAST-ACK
10. TIME-WAIT
11. CLOSED

LISTEN
: represents waiting for a connection request from any remote TCP and port. (usually set by TCP servers)

SYN-SENT
: represents waiting for the remote TCP to send back a TCP packet with the SYN and ACK flags set. (usually set by TCP clients)

SYN-RECEIVED
: represents waiting for the remote TCP to send back an acknowledgment after having sent back a connection acknowledgment to the remote TCP. (usually set by TCP servers)

ESTABLISHED
: represents that the port is ready to receive/send data from/to the remote TCP. (set by TCP clients and servers)

TIME-WAIT
: represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request. According to RFC 793 (http://tools.ietf.org/html/rfc793) a connection can stay in TIME-WAIT for a maximum of four minutes.

Connection establishment

To establish a connection, TCP uses a three-way handshake. Before a client attempts to connect with a server, the server must first bind to a port to open it up for connections: this is called a passive open. Once the passive open is established, a client may initiate an active open. To establish a connection, the

three-way (or 3-step) handshake occurs:

1. The active open is performed by the client sending a SYN to the server.
2. In response, the server replies with a SYN-ACK.
3. Finally the client sends an ACK back to the server.

At this point, both the client and server have received an acknowledgment of the connection.

Example:

1. The initiating host (client) sends a synchronization (SYN flag set) packet to initiate a connection. Any SYN packet holds a Sequence Number. The Sequence Number is a 32-bit field in TCP segment header. Let the Sequence Number value for this session be x.
2. The other host receives the packet, records the Sequence Number x from the client, and replies with an acknowledgment and synchronization (SYN-ACK). The Acknowledgment is a 32-bit field in TCP segment header. It contains the next sequence number that this host is expecting to receive ($x + 1$). The host also initiates a return session. This includes a TCP segment with its own initial Sequence Number of value y.
3. The initiating host responds with the next Sequence Number ($x + 1$) and a simple Acknowledgment Number value of $y + 1$, which is the Sequence Number value of the other host + 1.

Vulnerability to Denial of Service:
By using a spoofed IP address and repeatedly sending SYN packets attackers can cause the server to consume large amounts of resources keeping track of the bogus connections. Proposed solutions to this problem include SYN cookies and Cryptographic puzzles

Data transfer

There are a few key features that set TCP apart from User Datagram Protocol:

- Ordered data transfer
- Retransmission of lost packets
- Discarding duplicate packets
- Error-free data transfer
- Congestion/Flow control

Ordered data transfer, retransmission of lost packets and discarding duplicate packets

In the first two steps of the 3-way handshaking, both computers exchange an initial sequence number (ISN). This number can be arbitrary. This sequence number identifies the order of the bytes sent from each computer so that the data transferred is in order regardless of any fragmentation or disordering that occurs during transmission. For every byte transmitted the sequence number must be incremented.

Conceptually, each byte sent is assigned a sequence number and the receiver then sends an acknowledgment back to the sender that effectively states that they received it. What is done in practice is only the first data byte is assigned a sequence number which is inserted in the sequence number field and the receiver sends an acknowledgment value of the next byte they expect to receive.

For example, if computer A sends 4 bytes with a sequence number of 100 (conceptually, the four bytes would have a sequence number of 100, 101, 102, & 103 assigned) then the receiver would send back an acknowledgment of 104 since that is the next byte it expects to receive in the next packet. By sending an acknowledgment of 104, the receiver is signaling that it received bytes 100, 101, 102, & 103 correctly. If, by some chance, the last two bytes were corrupted then an acknowledgment value of 102 would be sent since 100 & 101 were received successfully.

However, a problem can occasionally arise when packets are lost. For example, 10,000 bytes are sent in 10 different TCP packets, and the first packet is lost during transmission. The sender would then have to resend all 10,000 bytes; the recipient cannot say that it received bytes 1,000 to 9,999 but only that it failed to receive the first packet, containing bytes 0 to 999. In order to solve this problem, an option of *selective acknowledgment (SACK)* has been added. This option allows the receiver to acknowledge isolated blocks of packets that were received correctly, rather than the sequence number of the last packet received successively, as in the basic TCP acknowledgment. Each block is conveyed by the starting and ending sequence numbers. In the example above, the receiver would send SACK with sequence numbers 1,000 and 10,000. The sender will thus retransmit only the first packet.

The SACK option is not mandatory and it is used only if both parties support it. This is negotiated when connection is established. SACK uses the optional part of the TCP header. See #TCP segment structure. The use of SACK is widespread - all popular TCP stacks support it. Selective acknowledgment is also used in SCTP.

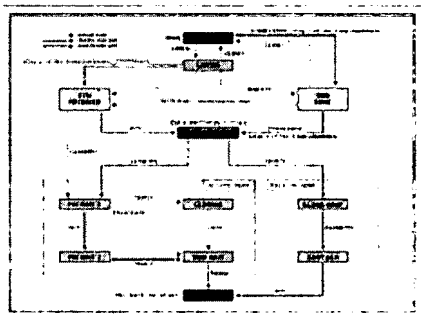
Error-free data transfer

Sequence numbers and acknowledgments cover discarding duplicate packets, retransmission of lost packets, and ordered-data transfer. To assure correctness a checksum field is included (*see TCP segment structure for details on checksumming*).

The TCP checksum is a quite weak check by modern standards. Data Link Layers with a high probability of bit error rates may require additional link error correction/detection capabilities. If TCP were to be redesigned today, it would most probably have a 32-bit cyclic redundancy check specified as an error check instead of the current checksum. The weak checksum is partially compensated for by the common use of a CRC or better integrity check at layer 2, below both TCP and IP, such as is used in PPP or the Ethernet frame. However, this does not mean that the 16-bit TCP checksum is redundant: remarkably, surveys of Internet traffic have shown that software and hardware errors that introduce errors in packets between CRC-protected hops are common, and that the end-to-end 16-bit TCP checksum catches most of these simple errors. This is the end-to-end principle at work.

Congestion control

The final part to TCP is congestion control. TCP uses a number of mechanisms to achieve high performance and avoid 'congestion collapse', where network performance can fall by several orders of magnitude. These mechanisms control the rate of data entering the network, keeping the data flow below a rate that would trigger collapse.



A Simplified TCP State Diagram. See *

TCP EFSM diagram (http://www.medianet.kent.edu/technicalreports.html#TR2005-07-22) for a more detailed state diagram including the states inside the ESTABLISHED state.

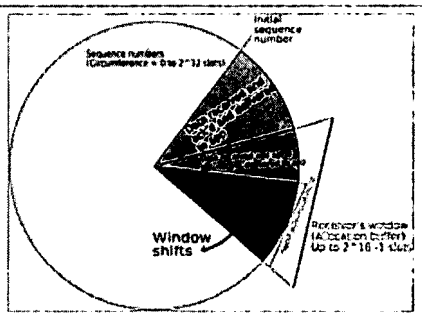
Acknowledgments for data sent, or lack of acknowledgments, are used by senders to implicitly interpret network conditions between the TCP sender and receiver. Coupled with timers, TCP senders and receivers can alter the behavior of the flow of data. This is more generally referred to as flow control, congestion control and/or network congestion avoidance.

Modern implementations of TCP contain four intertwined algorithms: Slow-start, congestion avoidance, fast retransmit, and fast recovery (RFC2581 (http://rfc.sunsite.dk/rfc/rfc2581.html)).

Enhancing TCP to reliably handle loss, minimize errors, manage congestion and go fast in very high-speed environments are ongoing areas of research and standards development.

TCP window size

The TCP receive window size is the amount of received data (in bytes) that can be buffered during a connection. The sending host can send only up to that amount of data before it must wait for an acknowledgment and window update from the receiving host. When a receiver advertises the window size of 0, the sender stops sending data and starts the persist timer. The persist timer is used to protect TCP from the dead lock situation. The dead lock situation could be when the new window size update from the receiver is lost and the receiver has no more data to send while the sender is waiting for the new window size update. When the persist timer expires the TCP sender sends a small packet so that the receivers ACKs the packet with the new window size and TCP can recover from such situations.



TCP sequence numbers and windows behave very much like a clock. The window, whose width (in bytes) is defined by the receiving host, shifts each time it receives and acks a segment of data. Once it runs out of sequence numbers, it loops back to 0.

Window scaling

For more efficient use of high bandwidth networks, a larger TCP window size may be used. The TCP window size field controls the flow of data and is limited to between 2 and 65,535 bytes.

Since the size field cannot be expanded, a scaling factor is used. The TCP window scale option, as defined in RFC 1323 (http://tools.ietf.org/html/rfc1323), is an option used to increase the maximum window size from 65,535 bytes to 1 Gigabyte. Scaling up to larger window sizes is a part of what is necessary for TCP Tuning.

The window scale option is used only during the TCP 3-way handshake. The window scale value represents the number of bits to left-shift the 16-bit window size field. The window scale value can be set from 0 (no shift) to 14.

Many routers and packet firewalls rewrite the window scaling factor during a transmission. This causes

sending and receiving sides to assume different TCP window sizes. The result is non-stable traffic that is very slow. The problem is visible on some sending and receiving sites which are behind the path of broken routers.

For more information on problems that may be caused, especially with Linux and Vista systems, please see main topic TCP window scale option.

Connection termination

The connection termination phase uses, at most, a four-way handshake, with each side of the connection terminating independently. When an endpoint wishes to stop its half of the connection, it transmits a FIN packet, which the other end acknowledges with an ACK. Therefore, a typical tear down requires a pair of FIN and ACK segments from each TCP endpoint.

A connection can be "half-open", in which case one side has terminated its end, but the other has not. The side that has terminated can no longer send any data into the connection, but the other side can.

It is also possible to terminate the connection by a 3-way handshake, when host A sends a FIN and host B replies with a FIN & ACK (merely combines 2 steps into one) and host A replies with an ACK. This is perhaps the most common method.

It is possible for both hosts to send FINs simultaneously then both just have to ACK. This could possibly be considered a 2-way handshake since the FIN/ACK sequence is done in parallel for both directions.

Some host TCP stacks may implement a "half-duplex" close sequence, as Linux or HP-UX do. If such a host actively closes a connection but still has not read all the incoming data the stack already received from the link, this host will send a RST instead of a FIN (Section 4.2.2.13 in RFC 1122 (http://tools.ietf.org/html/rfc1122)). This allows a TCP application to be sure that the remote application has read all the data the former sent - waiting the FIN from the remote side when it will actively close the connection. Unfortunatelly, the remote TCP stack cannot distinguish between a *Connection Aborting RST* and this *Data Loss RST* - both will make the remote stack to throw away all the data it received, but the application still didn't read.

Some application protocols may violate the OSI model layers, using the TCP open/close handshaking for the application protocol open/close handshaking - these may find the RST problem on active close. As an example:

```
s = connect(remote);
send(s, data);
close(s);
```

For a usual program flow like above, a TCP/IP stack like that described above does not guarantee that all the data will arrive to the other application *unless* the programmer is sure that the remote side will not send anything.

TCP ports

TCP uses the notion of port numbers to identify sending and receiving application end-points on a host, or *Internet sockets*. Each side of a TCP connection has an associated 16-bit unsigned port number (1-65535) reserved by the sending or receiving application. Arriving TCP data packets are identified as belonging to a specific TCP connection by its sockets, that is, the combination of source host address, source port, destination host address, and destination port. This means that a server computer can provide several clients with several services simultaneously, as long as a client takes care of initiating any simultaneous connections to one destination port from different source ports.

Port numbers are categorized into three basic categories: well-known, registered, and dynamic/private. The well-known ports are assigned by the Internet Assigned Numbers Authority (IANA) and are typically used by system-level or root processes. Well-known applications running as servers and passively listening for connections typically use these ports. Some examples include: FTP (21), ssh (22), TELNET (23), SMTP (25) and HTTP (80). Registered ports are typically used by end user applications as ephemeral source ports when contacting servers, but they can also identify named services that have been registered by a third party. Dynamic/private ports can also be used by end user applications, but are less commonly so. Dynamic/private ports do not contain any meaning outside of any particular TCP connection.

Development of TCP

TCP is a complex and evolving protocol. However, while significant enhancements have been made and proposed over the years, its most basic operation has not changed significantly since its first specification RFC 675 (http://www.ietf.org/rfc/rfc675.txt) in 1974, and the v4 specification RFC 793 (http://tools.ietf.org/html/rfc793), published in September 1981.[1] (http://www.faqs.org/rfcs/rfc793.html) RFC 1122 (http://tools.ietf.org/html/rfc1122), Host Requirements for Internet Hosts, clarified a number of TCP protocol implementation requirements. RFC 2581 (http://tools.ietf.org/html/rfc2581), TCP Congestion Control, one of the most important TCP related RFCs in recent years, describes updated algorithms to be used in order to avoid undue congestion. In 2001, RFC 3168 (http://tools.ietf.org/html/rfc3168) was written to describe explicit congestion notification (ECN), a congestion avoidance signalling mechanism. Common applications that use TCP include HTTP (World Wide Web), SMTP (e-mail) and FTP (file transfer).

The original TCP congestion avoidance algorithm was known as "TCP Tahoe", but many alternative algorithms have since been proposed.

TCP over wireless

TCP has been optimized for wired networks. Any packet loss is considered to be the result of congestion and the window size is reduced dramatically as a precaution. However, wireless links are known to experience sporadic and usually temporary losses due to fading, shadowing, hand off, etc. that cannot be considered congestion. Erroneous back-off of the window size due to wireless packet loss is followed by a congestion avoidance phase with a conservative decrease in window size which causes the radio link to be underutilized. Extensive research has been done on this subject on how to combat these harmful effects. Suggested solutions can be categorized as end-to-end solutions (which require modifications at the client and/or server), link layer solutions (such as RLP in CDMA2000), or proxy based solutions (which require some changes in the network without modifying end nodes).

Hardware TCP implementations

One way to overcome the processing power requirements of TCP is building hardware implementations of it, widely known as TCP Offload Engines (TOE). The main problem of TOEs is that they are hard to integrate into computing systems, requiring extensive changes in the operating system of the computer or device. The first company to develop such a device was Alacritech.

Debugging TCP

A packet sniffer, which intercepts TCP traffic on a network link, can be useful in debugging networks, network stacks and applications which use TCP by showing the user what packets are passing through a link. Some networking stacks support the SO_DEBUG socket option, which can be enabled on the socket using setsockopt. That option dumps all the packets, TCP states and events on that socket which will be helpful in debugging. netstat is another utility that can be used for debugging.

Alternatives to TCP

For many applications TCP is not appropriate. One big problem (at least with normal implementations) is that the application cannot get at the packets coming after a lost packet until the retransmitted copy of the lost packet is received. This causes problems for real-time applications such as streaming multimedia (such as Internet radio), real-time multiplayer games and voice over IP (VoIP) where it is sometimes more useful to get most of the data in a timely fashion than it is to get all of the data in order.

Also for embedded systems, network booting and servers that serve simple requests from huge numbers of clients (e.g. DNS servers) the complexity of TCP can be a problem. Finally some tricks such as transmitting data between two hosts that are both behind NAT (using STUN or similar systems) are far simpler without a relatively complex protocol like TCP in the way.

Generally where TCP is unsuitable the User Datagram Protocol (UDP) is used. This provides the application multiplexing and checksums that TCP does, but does not handle building streams or retransmission giving the application developer the ability to code those in a way suitable for the situation and/or to replace them with other methods like forward error correction or interpolation.

SCTP is another IP protocol that provides reliable stream oriented services not so dissimilar from TCP. It is newer and considerably more complex than TCP so has not yet seen widespread deployment, however it is especially designed to be used in situations where reliability and near-real-time considerations are important.

Venturi Transport Protocol (VTP) is a patented proprietary protocol that is designed to replace TCP transparently in order to overcome perceived inefficiencies related to wireless data transport.

TCP also has some issues in high bandwidth utilization environments. The TCP congestion avoidance algorithm works very well for ad-hoc environments where it is not known who will be sending data, but if the environment is predictable, a timing based protocol such as ATM can avoid the overhead of the retransmits that TCP needs.

TCP segment structure

A TCP segment consists of two sections:

- header
- data

The header consists of 11 fields, of which only 10 are required. The eleventh field is optional (pink background in table) and aptly named: options.

TCP Header

TCP Header

Bit offset	Bits 0–3	4–7	8–15	16–31
0	Source port			Destination port
32	Sequence number			
64	Acknowledgment number			
96	Data offset	Reserved	Flags	Window
128	Checksum			Urgent pointer
160	Options (optional)			
160/192+	Data			

- Source port – identifies the sending port
- Destination port – identifies the receiving port
- Sequence number – has a dual role
  - If the SYN flag is present then this is the initial sequence number and the first data byte is the sequence number plus 1
  - if the SYN flag is not present then the first data byte is the sequence number
- Acknowledgment number – if the ACK flag is set then the value of this field is the sequence number that the sender of the acknowledgment expects next.
- Data offset – specifies the size of the TCP header in 32-bit words. The minimum size header is 5 words and the maximum is 15 words thus giving the minimum size of 20 bytes and maximum of 60 bytes. This field gets its name from the fact that it is also the offset from the start of the TCP packet to the data.
- Reserved – for future use and should be set to zero
- Flags (aka Control bits) – contains 8 bit flags
  - CWR – Congestion Window Reduced (CWR) flag is set by the sending host to indicate that it received a TCP segment with the ECE flag set (added to header by RFC 3168 (http://tools.ietf.org/html/rfc3168)).
  - ECE (ECN-Echo) – indicate that the TCP peer is ECN capable during 3-way handshake (added to header by RFC 3168 (http://tools.ietf.org/html/rfc3168)).
  - URG – indicates that the URGent pointer field is significant

  - ACK – indicates that the ACKnowledgment field is significant
  - PSH – Push function
  - RST – Reset the connection
  - SYN – Synchronize sequence numbers
  - FIN – No more data from sender
- Window – the number of bytes that may be received on the receiving side before being halted from sliding any further and receiving any more bytes as a result of a packet at the beginning of the sliding window not having been acknowledged or received. Starts at acknowledgement field.
- Checksum – The 16-bit checksum field is used for error-checking of the header *and data*
- Urgent pointer – if the URG flag is set, then this 16-bit field is an offset from the sequence number indicating the last urgent data byte
- Options – the total length of the option field must be a multiple of a 32-bit word and the data offset field adjusted appropriately

Fields used to compute the checksum

TCP checksum using IPv4

When TCP runs over IPv4, the method used to compute the checksum is defined in RFC 793 (http://tools.ietf.org/html/rfc793):

> *The checksum field is the 16 bit one's complement of the one's complement sum of all 16-bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16-bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros.*

In other words, all 16-bit words are summed together using one's complement (with the checksum field set to zero). The sum is then one's complemented. This final value is then inserted as the checksum field. Algorithmically speaking, this is the same as for IPv6. The difference is in the data used to make the checksum. When computing the checksum, a pseudo-header that mimics the IPv4 header is shown in the table below.

TCP pseudo-header (IPv4)

Bit offset	Bits 0–3	4–7	8–15	16–31
0	Source address			
32	Destination address			
64	Zeros		Protocol	TCP length
96	Source port			Destination port
128	Sequence number			
160	Acknowledgement number			
192	Data offset	Reserved	Flags	Window
224	Checksum			Urgent pointer
256	Options (optional)			

256/288+	Data

The source and destination addresses are those in the IPv4 header. The protocol is that for TCP (*see List of IPv4 protocol numbers*): 6. The TCP length field is the length of the TCP header and data.

TCP checksum using IPv6

When TCP runs over IPv6, the method used to compute the checksum is changed, as per RFC 2460 (http://tools.ietf.org/html/rfc2460):

> *Any transport or other upper-layer protocol that includes the addresses from the IP header in its checksum computation must be modified for use over IPv6, to include the 128-bit IPv6 addresses instead of 32-bit IPv4 addresses.*

When computing the checksum, a pseudo-header that mimics the IPv6 header is shown in the table below.

TCP pseudo-header (IPv6)

Bit offset	**Bits 0 - 7**	**8–15**	**16–23**	**24–31**
0	Source address			
32				
64				
96				
128	Destination address			
160				
192				
224				
256	TCP length			
288	Zeros			Next header
320	Source port		Destination port	
352	Sequence number			
384	Acknowledgement number			
416	Data offset	Reserved	Flags	Window
448	Checksum		Urgent pointer	
480	Options (optional)			
480/512+	Data			

- Source address – the one in the IPv6 header
- Destination address – the final destination; if the IPv6 packet doesn't contain a Routing header, that will be the destination address in the IPv6 header, otherwise, at the originating node, it will be the address in the last element of the Routing header, and, at the receiving node, it will be the destination address in the IPv6 header.
- TCP length – the length of the TCP header and data;
- Next Header – the protocol value for TCP

Data

The last field is not a part of the header. The contents of this field are whatever the upper layer protocol wants but this protocol is not set in the header and is presumed based on the port selection.

See also

- Connection-oriented protocol
- T/TCP variant of TCP
- TCP and UDP port
- TCP and UDP port numbers for a complete (growing) list of ports/services
- TCP congestion avoidance algorithms
- TCP segment
- TCP Sequence Prediction Attack
- TCP Tuning for high performance networks
- Path MTU discovery
- SYN flood
- tcphdr - the Unix TCP header structure in the C programming language
- SCTP
- Transport protocol comparison table

References

1. ^ Comer, Douglas E. (2006). *Internetworking with TCP/IP:Principles, Protocols, and Architecture*, 5th, Prentice Hall.

External links

- TCP 3-Way Handshake Example (http://www.3wayhandshake.com/)
- Dissertation about TCP improvements in wired and wireless networks (http://edocs.tu-berlin.de/diss/2004/savoric_michael.htm) (Dissertation)
- IANA Port Assignments (http://www.iana.org/assignments/port-numbers)
- John Kristoff's Overview of TCP (Fundamental concepts behind TCP and how it is used to transport data between two endpoints) (http://condor.depaul.edu/~jkristof/technotes/tcp.html)
- RFC 675 (http://tools.ietf.org/html/rfc675) - Specification of Internet Transmission Control Program, December 1974 Version
- RFC 793 (http://tools.ietf.org/html/rfc793) - TCP v4
- RFC 1122 (http://tools.ietf.org/html/rfc1122) - some error-corrections
- RFC 1323 (http://tools.ietf.org/html/rfc1323) - TCP-Extensions
- RFC 2581 (http://tools.ietf.org/html/rfc2581) - TCP Congestion Control
- RFC 4614 (http://tools.ietf.org/html/rfc4614) - A Roadmap for TCP Specification Documents

- TCP, Transmission Control Protocol (http://www.networksorcery.com/enp/protocol/tcp.htm)
- TCP EFSM diagram - A detailed description of TCP states. (http://www.medianet.kent.edu/technicalreports.html#TR2005-07-22)
- The basics of Transmission Control Protocol (http://tcp.mywebcities.com/)
- The Law of Leaky Abstractions (http://www.joelonsoftware.com/articles/LeakyAbstractions.html) by Joel Spolsky
- Checksum example (http://mathforum.org/library/drmath/view/54379.html)

Retrieved from "http://en.wikipedia.org/wiki/Transmission_Control_Protocol"

Categories: Cleanup from August 2007 | Wikipedia articles needing clarification | TCP/IP | Transport layer protocols